Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Gobov

Določanje najkrajših poti med vozlišči telekomunikacijskega omrežja

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi opišite rešitev problema iskanja najkrajših poti med vsemi točkami telekomunikacijskega omrežja, ki je predstavljeno z neusmerjenim grafom. V ta namen predstavite opis omrežja v ustreznem formatu, izberite najprimernejši algoritem ter operativno realizirajte celotno izvedbo iskanja in primeren grafični prikaz rešitve.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miha Gobov, z vpisno številko **63050472**, sem avtor diplomskega dela z naslovom:

Določanje najkrajših poti med vozlišči telekomunikacijskega omrežja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Igorja Rožanca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 20. junij 2014 Podpis avtorja:

Kazalo

Povzetek

Diplomsko delo opisuje rešitev problema iskanja povezav med točkama v telekomunikacijskem omrežju. Za rešitev problema imamo na voljo podroben prostorski izris omrežja ter seznam parov začetnih in končnih koordinat, med katerimi je treba poiskati najkrajše poti.

Podatki o omrežju so podani v formatu shapefile, ki je namenjen shranjevanju vektorskih podatkov v geografskih informacijskih sistemih. Da bi lahko iskali poti v podanem omrežju, smo ga pretvorili v obliko grafa. Tega smo shranili v grafno podatkovno bazo, to je namensko orodje za delo z grafi v obliki vozlišč in povezav. Začetne in končne koordinate smo nato povezali na liste v grafu. Za iskanje najkrajših poti smo uporabili Dijkstrov algoritem. Rezultat smo nato shranili v obliko CSV. Rešitev smo vizualno prikazali tudi v geografskem informacijskem sistemu, kjer smo za uvoz podatkov uporabili obliko Well-known text. Izbran pristop se je izkazal za učinkovitega tako s stališča enostavnosti implementacije kot tudi hitrosti izvajanja programa.

Ključne besede: prostorski podatki, grafne podatkovne baze, problem iskanja najkrajših poti, Dijkstrin algoritem iskanja najkrajših poti

Abstract

The thesis describes a solution to the shortest path problem between two points in a telecommunication network. To solve the problem a detailed spatial data describing network and the list of start and end coordinate pairs are available, between which we have to find the shortest path.

The network layout is provided in the shapefile format which is used for storing vector geospatial data in geographic information systems. In order to be able to search for paths in the network, we have transformed it into a graph form. We have saved the graph into the graph database, which is a purpose tool for working with graphs in the form of vertices and edges. Start and end coordinate points were then connected to the leaves in the graph. In order to look for of the shortest paths we have used Dijkstra's algorithm. Results were then stored in the CSV format. The results were also visualised in the geographic information system, where we used the Well-known text format to import them. The chosen approach turned out to be effective from the simplicity of implementation and from the performance points of view.

Keywords: spatial data, graph database, shortest path problem, Dijkstra's algorithm for finding the shortest path

Poglavje 1

Uvod

V diplomski nalogi bomo reševali konkreten problem, s katerim se telekomunikacijsko podjetje srečuje v praksi. Inženirji na podlagi različnih analiz načrtujejo izpopolnjevanje telekomunikacijskega omrežja. Ena od takšnih analiz je obremenitev posameznih vodnikov v omrežju. Da bi lahko predvideli obremenitev posameznega vodnika v omrežju, morajo vedeti, kateri vodniki so uporabljeni za priklop uporabnika na centralo. Ker je na trgu trenutno potreba po takšni aplikaciji, smo se odločili, da jo razvijemo.

Geografski opis omrežja bomo najprej pretvorili v obliko grafa, da bomo lahko z uporabo algoritmov za iskanje najkrajših poti poiskali povezave med uporabniki in centralami. Ker pa podatki o omrežju in podatki o lokacijah uporabnikov niso popolni, si bomo pomagali s hevurističnimi pravili, s katerimi bomo popravili in nadomestili pomanjkljivosti podatkov, in s tem izboljšali rezultate.

V vsebini diplomskega dela se bomo dotaknili osnov geografskih informacijskih sistemov za lažje razumevanje oblike podatkov o omrežju. Spoznali bomo grafne podatkovne baze, ki jih bomo uporabili za delo z omrežjem, ko bo le-tega preurejamo v obliko grafa. Analizirali bomo vhodne podatke in zahteve uporabnikov ter na podlagi tega naredili načrt aplikacije. Na koncu bomo predstavili implementacijo rešitve in povzeli uspešnost delovanja končne aplikacije v praksi.

Poglavje 2

Geografski informacijski sistemi in grafne podatkovne baze

Geografski informacijski sistem [1] (angl. Geographic Information System) ali krajše GIS je zbirka orodij, ki uporabnikom omogoča naslednje operacije nad prostorskimi podatki:

- zbiranje,
- shranjevanje,
- iskanje,
- prikazovanje in
- pretvorbe.

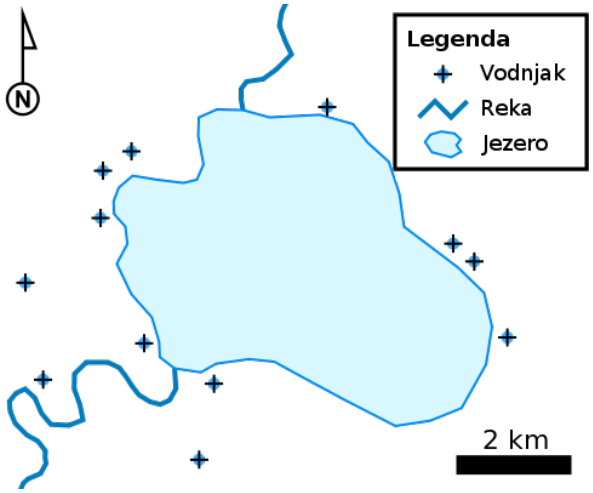
Storitev GIS je navadno sestavljena iz podatkovne baze prostorskih podatkov in nabora podpornih aplikacij ali aplikacijskih modulov, preko katerih poteka interakcija med uporabnikom in podatki.

GIS se vedno pogosteje znajdejo v naboru orodij v mnogih poklicih, pa tudi pri uporabi v prostem času. Skoraj vsak od nas, se je že kdaj v življenju srečal s katerim od takšnih orodij. Med drugim spadajo med orodja GIS tudi Google Maps ali satelitska navigacija na pametnih telefonih in v avtomobilih.

2.1 Prostorski podatki

Prostorski podatki v aplikacijah GIS so predstavljeni v dveh osnovnih oblikah: v vektorski ali rasterski obliki.

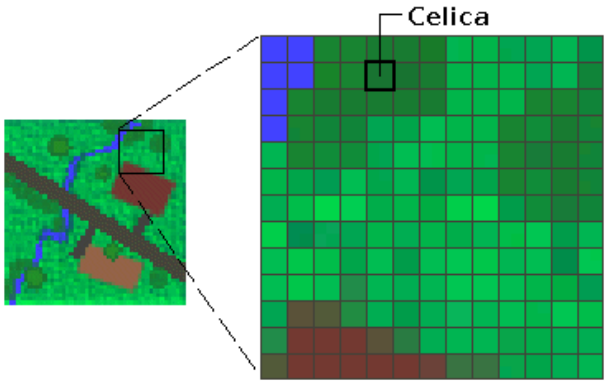
Vektorski podatki zajemajo točke, črte, poligone in območja. Uporabni so za podatke, ki imajo diskretne meje, kot so meje držav, parcel in ulice. Slika 2.1 je prikaz uporabe vektorskih podatkov. Točke prikazujejo lokacije vodnjakov, temno modre črte potek reke, območje, pobarvano s svetlo modro ter obrobljeno s temnejšo modro, pa nakazuje jezero.



Slika 2.1: Primer vektorskih prostorskih podatkov.

Rasterski podatki so sestavljeni iz matrik celic, organiziranih v vrstice in stolpce, kot je prikazano na sliki 2.2. Vsaka celica predstavlja informacijo o določeni lokaciji. Rasterski podatki se uporabljajo za predstavitev digitalnih letalskih posnetkov, satelitskih posnetkov, skeniranih zemljevidov, temperatur, itd.

Vsaka smiselna skupina podatkov se navadno shrani v svojo plast (angl. layer). Takšne plasti je potem z aplikacijami GIS mogoče pregledovati posamično ali eno nad drugo. Slika 2.3 v zgornji polovici prikazuje satelitsko sliko, predstavljeno z rastersko plastjo. V spodnjem delu pa sta nad ob-



Slika 2.2: Primer rasterskih prostorskih podatkov.

stoječo rastersko dodani še dve vektorski plasti. Prva plast prikazuje meje držav, druga plast pa lokacije večjih mest ter njihova imena.

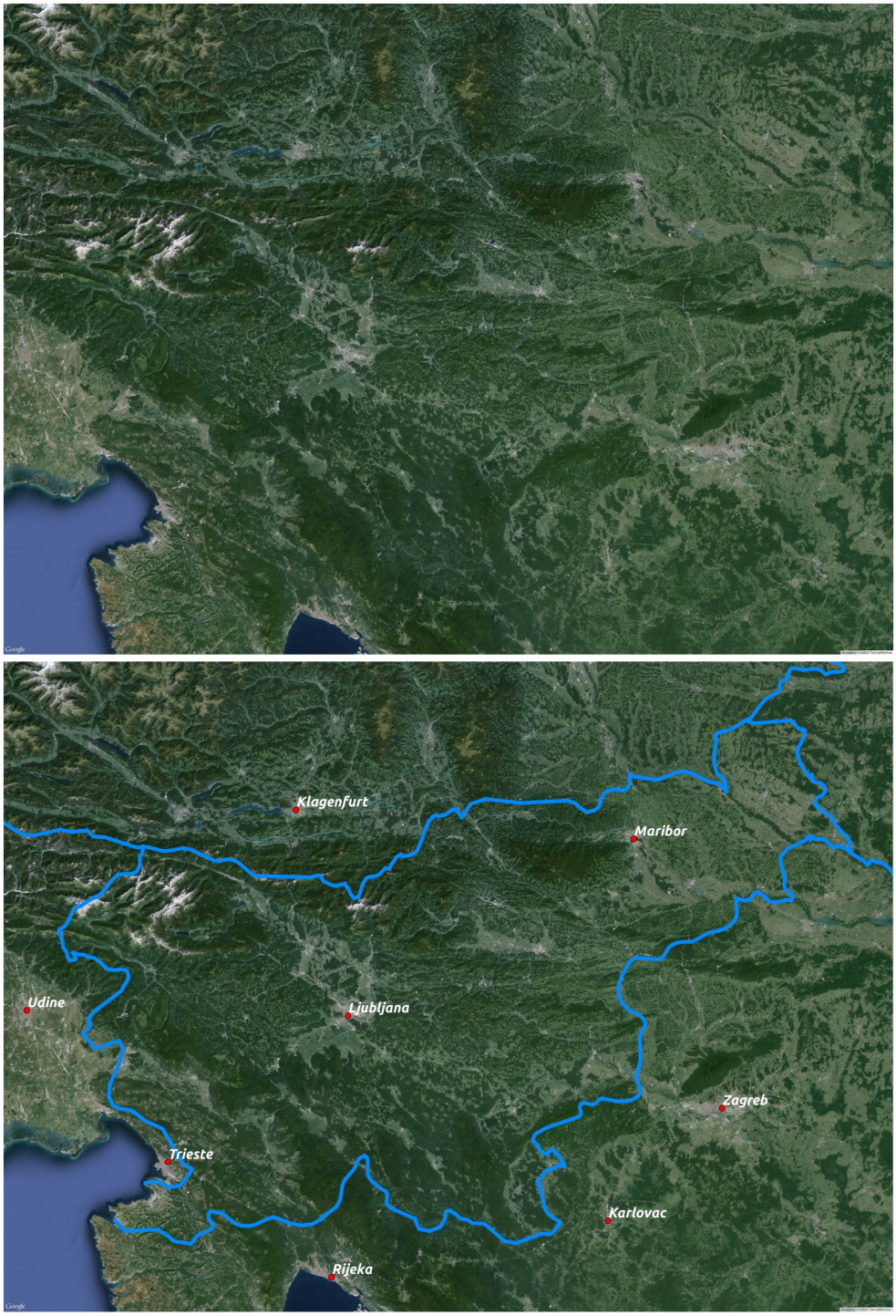
2.2 Neprostorski podatki

Poleg prostorskih podatkov je včasih treba shraniti tudi neprostorske podatke, ki so vezani na vektorski element ali rastersko celico. Uporabljajo se za shranjevanje lastnosti elementa, na katerega se nanašajo. Slika 2.3 prikazuje plast z lokacijami mest z rdečimi pikami. Na te prostorske podatke je vezana še lastnost, ki hrani imena mest. Ta lastnost je prikazana ob vsakem prostorskem elementu kot napis bele barve.

2.3 Format shapefile

Shapefile je ena od oblik shranjevanja vektorskih prostorskih podatkov. Razvili so ga pri Esri [14] v zgodnjih devetdesetih letih in ga predstavili tržišču kot odprti standard (angl. open standard).

Naziv shapefile daje mogoče napačen vtis, da gre le za eno datoteko, čeprav gre za skupek datotek [16]. Podatki so shranjeni v treh ali več tipih datotek. Pri datotekah se uporablja enotno poimenovanje (angl. naming



Slika 2.3: Prikaz uporabe plasti [12, 13].

convention). Datoteke, ki predstavljajo en shapefile, morajo imeti enako predpono, pripona pa določa, za kateri tip datoteke gre. Trije obvezni tipi datotek so:

- **shape format** (pripona `.shp`) hrani geometrijske elemente,
- **shape index format** (pripona `.shx`) hrani pozicijski indeks geometrijskih elementov, kar omogoča hitro iskanje naprej in nazaj ter
- **attribute format** (pripona `.dbf`) hrani stolpično shranjene lastnosti elementov v dBase IV obliki [15].

Ostali tipi datotek se uporabljajo za shranjevanje podatkov, kot so opis projekcije oz. tip koordinatnega sistema, prostorskih indeksov, itd.

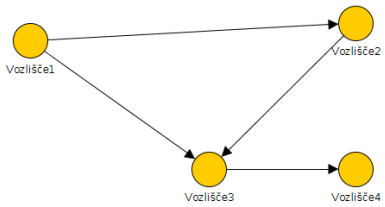
Ker je shapefile uveljavljen odprti standard, podpira delo z njim večina aplikacij GIS. Na voljo je tudi širok nabor programskih knjižnic za različne programske jezike.

2.4 Graf kot matematična struktura

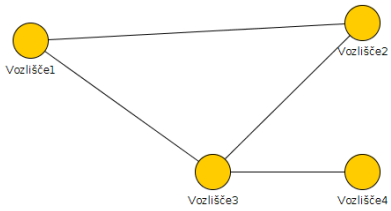
Graf je struktura v diskretni matematiki, s katero lahko ponazorimo omrežje (cest, železnic, svetovnega spleta, sistema kanalov, molekul, ...). Sestavljen je iz vozlišč oz. točk, ki lahko predstavljajo kraje, postaje, računalnike, atome, in povezav, ki opisujejo ceste, žice, kanale, vezi in lahko nosijo različne lastnosti [7]. Matematično disciplino, ki preučuje lastnosti grafov, imenujemo teorija grafov.

Grafe delimo na usmerjene in neusmerjene. Pri usmerjenih grafih ima vsaka povezava določeno tudi smer. Slika 2.4 prikazuje usmerjen graf, puščice na povezavah pa nakazujejo smer povezave. Kadar smer ni določena, govorimo o neusmerjenih grafih. Slika 2.5 prikazuje primer neusmerjenega grafa.

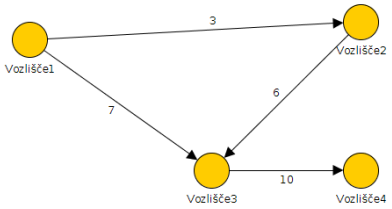
Če povezavam pripišemo tudi uteži, nastane utežen graf. Utež predstavlja lastnost povezave ali vozlišča, ki je navadno izražena kot pozitivno število in igra pomembno vlogo pri obhodih grafov. Če na primer graf predstavlja



Slika 2.4: Usmerjen graf.



Slika 2.5: Neusmerjen graf.



Slika 2.6: Usmerjen graf z utežmi na povezavah.

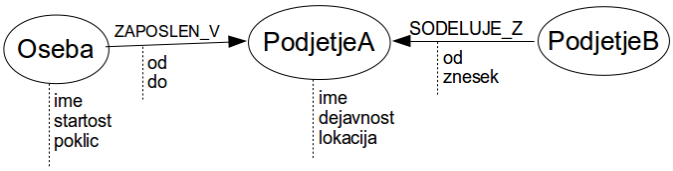
mrežo cest, potem lahko uteži na povezavah predstavljajo razdaljo ali čas, ki ga potrebujemo, da prepotujemo od ene do druge točke. Na sliki 2.6 lahko vidimo usmerjen graf z utežmi na povezavah.

2.5 Grafne podatkovne baze

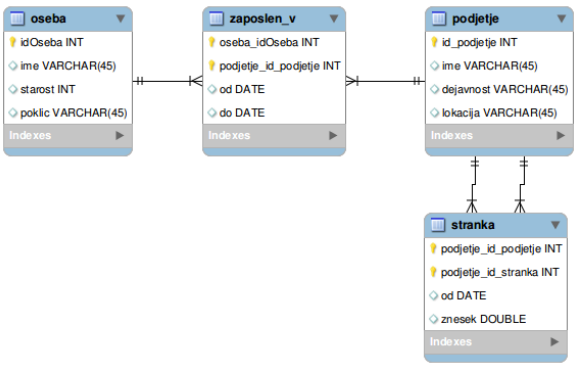
Grafne podatkovne baze so podatkovne baze, ki za modeliranje, shranjevanje in poizvedbe uporabljajo strukturo grafov z vozlišči in povezavami, ki jim lahko pripnemo tudi lastnosti. Uvrščajo se v skupino NoSQL[1] podatkovnih baz.

Značilnost grafnih podatkovnih baz je brezindeksno sosedstvo (angl. index-free adjacency). To pomeni, da ima vsak element kazalce na sosednje ele-

[1]NoSQL ali Not Only SQL podatkovne baze so mehanizmi shranjevanja in pridobivanja podatkov, ki za shranjevanje ne uporabljajo relacijskih tabel, kot jih uporabljajo relacijske podatkovne baze. [11]



Slika 2.7: Primer poslovnega modela.



Slika 2.8: Relacijski podatkovni model.

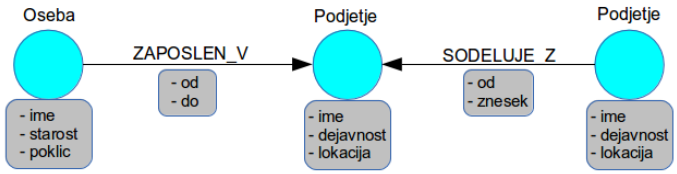
mente, do katerih je mogoče dostopati brez poizvedb z uporabo indeksov.

Ne gre podcenjevati tudi dejstva, da je lahko modeliranje določene domene z grafnimi podatkovnimi bazami bolj razumljivo. Strukturo modela grafne podatkovne baze je z risanjem lažje razložiti sodelavcem na projektu, ki nimajo poglobljenega znanja na področju podatkovnih baz. Naslednji primer prikazuje takšno situacijo.

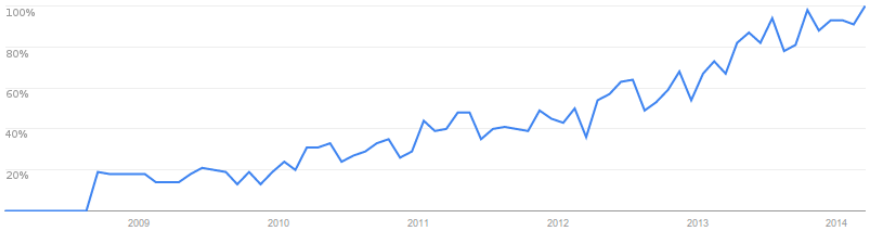
Modeliranje enostavnega problema [8] z relacijskimi in grafnimi podatkovnimi bazami je lepo prikazano na sliki 2.7. Prikazan je problem, ki je narisan na belo tablo, kar je tipično ena začetnih faz razvoja projekta.

V naslednjem koraku je na sliki 2.8 prikazan prej omenjen problem, modeliran za relacijske podatkovne baze. Model je očitno težje razumljiv kot tedaj, ko je bil narisan na beli tabli. Za nekoga, ki se še ni srečal z relacijskimi podatkovnimi bazami, je tudi težko razumljiv.

Slika 2.9 prikazuje prej omenjeni problem, modeliran za grafne podat-



Slika 2.9: Grafni podatkovni model



Slika 2.10: Graf prikazuje trend besedne zveze "graph database" od leta 2008 naprej. X os je čas, Y os pa je trend, relativno na najvišjo točko v grafu [4].

kovne baze. Ta je zelo podoben modelu, ki je bil zasnovan na beli tabli. Lažje je razumljiv tudi tistim, ki nimajo izkušenj s podatkovnimi bazami. Olajšano je tudi delo načrtovalcem podatkovne baze, kar je očitno, če primerjamo sliki 2.7 in 2.9.

Večina trenutnih implementacij grafnih podatkovnih baz ne uporablja fiksnih shem podatkovnega modela, ki so značilne za relacijske baze. Pozitivna stran te lastnosti je, da so takšne baze bolj prilagodljive. Negativna stran pa je, da je naloga programerja, da pazi na integriteto podatkov pri spreminjanju baze.

V zadnjih letih je v porastu zanimanje za grafne podatkovne baze, kar nakazuje tudi analiza trendov iskalnika Google, ki jo prikazuje slika 2.10. Vse več podjetij se pri svojih ključnih produktih zanaša nanje, posledično raste tudi ponudba na trgu grafnih podatkovnih baz.

Ko se odločamo, katero od implementacij izbrati, se osredotočimo na dva kriterija [2]:

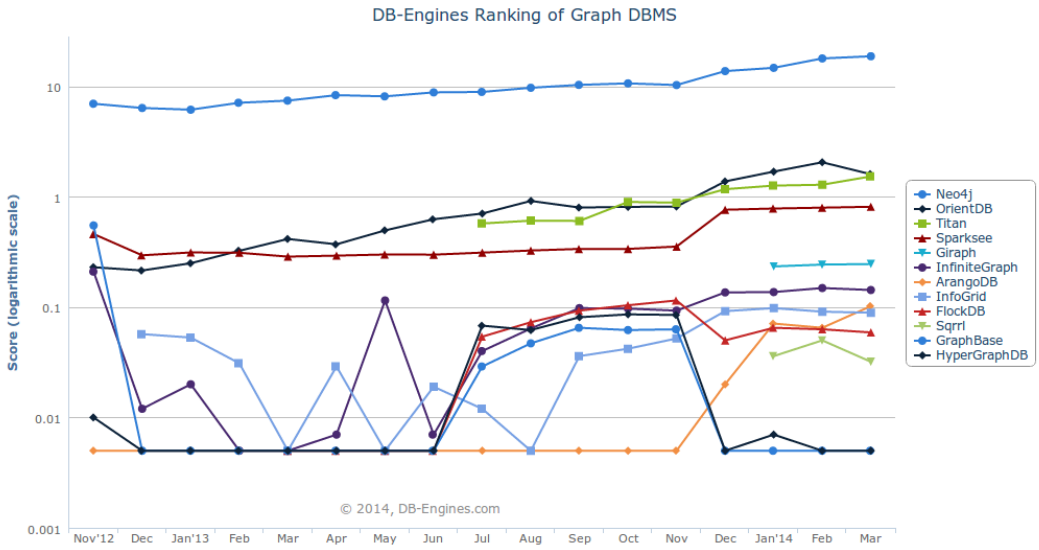
- **Način osnovnega shranjevanja (angl. the underlying storage)**
  Grafne podatkovne baze lahko uporabljajo avtohtoni (angl. native) način shranjevanja grafov, ki je optimiziran in uporablja grafne podatkovne strukture pri delu z grafi. Avtohtonega shranjevanja grafov ne uporabljajo vse implementacije. Nekatere uporabljajo serializacijo podatkov grafa v relacijske tabele, objektno usmerjene podatkovne baze ali katero od drugih oblik splošno namenskih podatkovnih skladišč.

- **Procesni mehanizem (angl. the processing engine)**
  Za grafne podatkovne baze je značilno, da uporabljajo brezindeksno sosedstvo, vendar lahko za grafne podatkovne baze štejemo vse, ki se navzven obnašajo kot grafne podatkovne baze, tudi če ne uporabljajo brezindeksnega sosedstva. Brezindeksno sosedstvo ima veliko zmogljivostno prednost pred ostalimi metodami, zato takšnim mehanizmom, ki to izkoriščajo, pravimo avtohtoni procesni mehanizmi.

Ker zmogljivostno največ obetajo grafne podatkovne baze z avtohtonim načinom shranjevanja in avtohtonim procesnim mehanizmom, smo se za potrebe diplomskega dela osredotočili ravno na te. Poleg omenjenih zahtev, smo pri izbiri upoštevali tudi naslednje kriterije:

- popularnost,
- ustrezno dokumentacijo,
- zrelost projekta in
- dostopnost brezplačne različice.

Za najbolj primernega se je izkazal **Neo4j**, ki je produkt podjetja Neo Technology [9]. Iz analize popularnosti grafnih podatkovnih baz na sliki 2.11 je razvidno, da se trenutno daleč najpogosteje uporablja ravno Neo4j [3].



Slika 2.11: Graf trendov priljubljenosti grafnih podatkovnih baz [3].

2.6 Sistem za upravljanje grafnih podatkovnih baz Neo4j

Neo4j je odprtokodni sistem za upravljanje grafnih podatkovnih baz [5]. V celoti je implementiran v Javi. Razvijalci ga opisujejo kot vgrajen (angl. embedded) transakcijski mehanizem, ki podatke trajnostno shranjuje na disku, strukturirane v grafe namesto v tabele.

Aktualna verzija ima omejeno število vozlišč, povezav in lastnosti. Razlog za omejitev je širina naslovnega prostora primarnih ključev. Za vozlišča in povezave se namreč uporablja 35 bitov, za lastnosti pa od 36 do 38 bitov, odvisno od podatkovnega tipa lastnosti. Tabela 2.1 prikazuje izračune, koliko posameznih elementov je mogoče shraniti v vsako Neo4j podatkovno bazo. Za naše potrebe omejitve ne predstavljajo problema, saj bomo delali s približno milijon vozlišči in povezavami.

V [2] je predstavljen eksperiment, ki prikazuje prednost grafnih podatkovnih baz, v primerjavi z relacijskimi bazami pri močno povezanih podat-

vozlišča	2^{35} (˜34 milijard)
povezave	2^{35} (˜34 milijard)
lastnosti	od 2^{36} do 2^{38}, odvisno od podatkovih tipov lastnosti (max.˜274 milijard, vedno več kot ˜68 milijard)
tipi povezav	2^{15} (˜32.000)

Tabela 2.1: Prikazuje maksimalno število posameznih elementov v podatkovni bazi Neo4j, v2.0.1.

Globina	RDBMS čas (s)	Neo4j čas (s)	Št. vrnjenih elementov
2	0,016	0,010	2.500
3	30,267	0,168	110.000
4	1.543,505	1,359	600.000
5	nedokončano	2,132	800.000

Tabela 2.2: Prikaz meritev eksperimenta na socialnem omrežju.

kih. Za eksperiment uporabijo primer socialne mreže 1.000.000 ljudi, kjer ima vsak približno 50 prijateljev. Naloga je najti vse prijatelje prijateljev, in sicer od globine 2 do 5. Iz tabele 2.2 je razvidno, da sta pri globini 2 oba sistema primerljiva, medtem ko se že pri globini 3 pojavi velika razlika in relacijske podatkovne baze postanejo neuporabne za spletno uporabo. Odzivni čas Neo4j kaže, da je tudi pri globini 4 še vedno v mejah za odzivne aplikacije.

2.6.1 Poizvedbeni jezik Cypher

Cypher je deklarativen grafni poizvedbeni jezik, ki je prav tako delo razvijalcev Neo4j [6]. Uporablja se za poizvedovanje in urejanje grafov. Sintaksa jezika spominja na SQL, predvsem zaradi podobnosti nekaterih stavkov oz. ključnih besed. Primer je stavek `WHERE`, ki ima podobno uporabo v obeh jezikih.

Enostavna poizvedba v jeziku Cypher se zgradi kot:

```
1 MATCH (a:Oseba)-[:Pozna]->(b:Oseba)
2 WHERE a.ime = 'Janez'
3 RETURN b AS znanci
```

s katero poiščemo vse osebe, ki jih pozna Janez. Razlago potrebuje predvsem prva vrstica s ključno besedo `MATCH`. Omenjen stavek je namenjen iskanju vzorca v grafu. Vzorec se oriše z vozlišči in povezavami. Vozlišča so predstavljena v okroglih oklepajih, povezave pa v oglatih oklepajih. Uporabil sem izraz oriše, ker lahko z malo domišljije bralec vidi podobnost med tekstovnim opisom `(a:Oseba)-[:Pozna]->(b:Oseba)` in grafično predstavitvijo na sliki 2.12.

Slika 2.12: Grafično predstavljena povezava.

Besedi za dvopičjem `Oseba` in `Pozna` opisujeta oznako vozlišča oz. tip povezave. Vsakemu vozlišču in povezavi lahko v Neo4j pripišemo oznako oz. tip, kar pripomore k urejenosti. Tako je recimo lažje ločevati med vozlišči, ki predstavljajo osebe, in tistimi, ki predstavljajo hišne ljubljenčke.

Snovalci jezika so si kot cilj zastavili tudi, da bi bil jezik razumljiv tako razvijalcem kot tudi laikom. Želeli so narediti preproste stvari enostavne in kompleksne stvari mogoče. Konstrukti so zasnovani na angleški prozi in urejeni ikonografiji, ki pomaga narediti poizvedbe samoumevne. Poizkusili so optimizirati jezik za branje in ne za pisanje [10].

Primeri malo zapletenejših poizvedb za konkretne probleme bodo predstavljeni v nadaljevanju.

Poglavje 3

Analiza podatkov in načrtovanje rešitve

Aplikacija mora sprejeti dve skupini vhodnih podatkov: prva je skupina s prostorskimi podatki, ki podrobno opisuje segmente vodnikov omrežja, druga skupina pa vsebuje pare točk A in Z, kjer točka A predstavlja centralo v omrežju, točka Z pa predstavlja končnega uporabnika. Naloga aplikacije je, da najprej za vsak A in Z, ki sta podana s koordinatami, poišče, kje se priključita na omrežje, nato pa poišče najkrajšo pot v omrežju, po kateri je mogoče priti od točke A do točke Z. Na koncu je treba izpisati seznam uporabljenih segmentov omrežja na poti.

3.1 Analiza vhodnih podatkov

Vhodne podatke smo dobili brez podrobne dokumentacije. Podatki o omrežju so bili shranjeni v obliki shapefile, tisti, ki so opisovali pare, pa v obliki CSV[1]. Da bi podrobneje spoznali, kaj določeni podatki predstavljajo in kako nam bodo koristili za dosego cilja, smo se lotili analize.

[1]Datoteka Comma-Separated Values ali krajše CSV je namenjena za shranjevanje tabelaričnih podatkov v tekstovni obliki.

Slika 3.1: Delček uvoženega omrežja

3.1.1 Omrežje

Kot je že bilo omenjeno v uvodu, so nam bili podatki o omrežju podani v obliki shapefile. Za pregled podanih podatkov smo uporabili odprtokodno orodje Quantum GIS[2] (krajše QGIS) [17]. Zanimalo nas je, v kakšni obliki so geografsko predstavljeni podatki ter katere lastnosti imajo elementi.

Po uvozu podatkov smo ugotovili, da so elementi podani s črtami, kar je bilo tudi pričakovano. Črte prikazujejo segmente vodnikov (paric), ki sestavljajo omrežje. S tem smo dobili odgovor na vprašanje v prvi alineji zgoraj. Izkazalo se je tudi, da je nabor vseh podatkov omrežja za pregledovanje v aplikaciji QGIS prehud zalogaj na povprečno zmogljivi strojni opremi. Razlog za to je velika količina podatkov. Kadar je bil v pogledu zajet večji del omrežja, je bilo izrisovanje moteče počasno, izris je trajal tudi več kot 15 sekund. Pri približanem pogledu, ko je bilo zajeto manjše območje omrežja, je bila odzivnost bistveno boljša. Takšno manjše območje, ki prikazuje vodnike delčka omrežja, je vidno na sliki 3.1.

Nato smo poiskali nabor lastnosti elementov. Tudi ta korak je bil z uporabo QGISa enostaven, saj omogoča tabelaričen pregled lastnosti uvoženih elementov. Stolpci v tabeli predstavljajo različne lastnosti, vsaka vrstica pa predstavlja element shapefilea – v našem primeru segment vodnika. Upora-

[2]QGIS je samostojna aplikacija, ki deluje na različnih operacijskih sistemih in omogoča pregledovanje, urejanje in analizo geografskih prostorskih podatkov.

bljen shapefile ni bil namenjen samo reševanju našega konkretnega problema in je zato vseboval tudi lastnosti, ki jih nismo potrebovali. Našli pa smo lastnost, ki je enolično predstavljala ime segmenta, kar je potrebno za opis poti od točke A do Z.

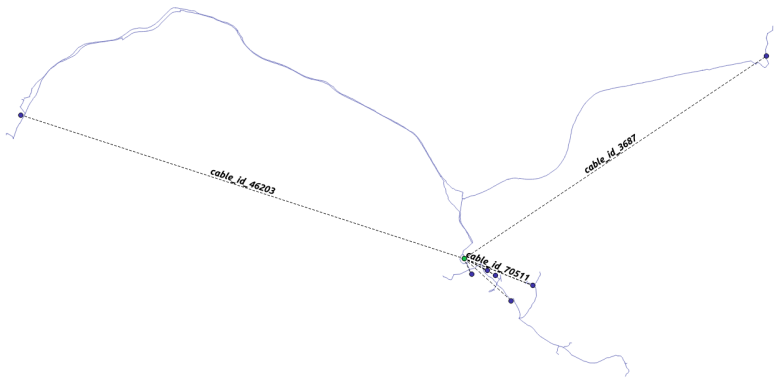
3.1.2 Datoteka parov točk koordinat central in uporabnikov

Datoteka vsebuje podatke, podane v obliki CSV. V stolpcih so podatki naslednji:

- enolična identifikacija para,
- x in y koordinati točke A, ki predstavlja centralo, in
- x in y koordinati točke Z, ki predstavlja uporabnika.

Za boljšo predstavo, kaj ti podatki v tekstovni obliki pomenijo v korelaciji z omrežjem, smo se odločili za dodaten korak. Zanimalo nas je, kako se podatki iz podane datoteke parov skladajo s predhodno uvoženimi podatki omrežja. QGIS sicer omogoča uvoz datotek CSV, vendar moramo datoteko razdeliti na tri podmnožice, da bi lahko pregledovali točke A (centrale), točke Z (uporabnike) na različnih plasteh ter jih povezali v pare. Vsako podmnožico smo shranili v ločeno CSV datoteko. Datoteki za točke A in Z sta vsebovali samo koordinate točk, za tretjo plast pa smo pripravili ravne črte med pari točk A in Z, da bodo na zemljevidu jasno prikazane povezave med točkami. Rezultat je viden na sliki 3.2, kjer smo nad plast omrežja s slike 3.1 dodali še na novo uvožene plasti. Plast z zelenimi pikami prikazuje točke A (centrale), plast z modrimi pikami točke Z (uporabnike), plast s prekinjenimi črtami pa povezuje pare. Na zadnji plasti je prikazana tudi črtam pripisana lastnost, ki predstavlja enolično identifikacijo para, če je prostor to omogočal.

Iz grafičnega pregleda je mogoče razbrati, da se novo uvožene plasti skladajo s predhodno uvoženim omrežjem, saj je lokacija točk vedno v bližini tras, gostota točk pa se ujema z gostoto vodnikov v omrežju. V kontekstu to



Slika 3.2: Omrežje z dodanimi plastmi za točke A in Z ter povezavami med njimi.

pomeni, da so v urbanih območjih vodniki gosteje posejani, tam je tudi več točk, medtem ko je v ruralnih območjih manj vodnikov in tudi manj točk.

3.2 Načrtovanje rešitve

Na podlagi podanih zahtev uporabnikov in ugotovitev glede vhodnih podatkov lahko začnemo načrtovati aplikacijo. Dober načrt aplikacije je zelo pomembna faza razvoja aplikacije, ki se ji v praksi dostikrat ne posveča dovolj pozornosti. Slabo načrtovanje lahko privede do velikih zapletov med implementacijo, zaradi česar lahko trpi kvaliteta končne rešitve in se zgodijo zamude predvidenih rokov izvedbe.

3.2.1 Pretvorba omrežja v graf

Omrežje v obliki shapefile je opisano s črtami, ki predstavljajo segmente omrežja. Prikazani so tudi vsi zavoji segmenta, kar pa ne vpliva na rešitev. Za rešitev nas zanima samo dolžina segmenta in povezanost z drugimi segmenti. Za naš primer bi torej lahko poenostavili omrežje kot skupek povezav, ki predstavljajo segmente, pri čemer ima vsaka povezava 2 lastnosti: enolični identifikator segmenta in dolžino v metrih.

Da lahko iz vhodnih podatkov sestavimo graf, je treba definirati vozlišča in povezave. Povezave so v prejšnjem odstavku že definirane, vendar je treba določiti še vozlišča. V primeru podanega omrežja, lahko za vozlišča uporabimo geografske točke, ki predstavljajo začetke in konce segmentov. Enolično jih določimo s koordinatami – vsakemu vozlišču bomo pripisali tudi lastnosti x in y.

3.2.2 Povezovanje točk A in Z z grafom omrežja

V analizi vhodnih podatkov je bilo ugotovljeno, da se točke A in Z v večini primerov ne dotikajo omrežja. Razloga za to sta dva:

- omrežje ni izrisano povsem do stanovanjskih oz. poslovnih objektov, temveč le do telekomunikacijskih priključnih omaric;
- koordinati točk A in Z sta določeni glede na podatek, kje se nahaja hišna številka. To pomeni, da je celoten objekt, ne glede na obliko in velikost, definiran kot ena sama točka. To prinese dodatno odstopanje v natančnosti.

Po posvetu s strokovnjaki iz podjetja smo določili hevristična pravila, ki določajo, kako se točke A in Z povežejo na omrežje:

1. Točke se povezujejo samo na liste grafa, ker navadno ravno list grafa določa mejne točke omrežja. V praksi to pomeni, da se vodnik, ki vodi do omarice ali centrale, tam zaključi.
2. Razdalja med točko in listom, na katerega jo povežemo, ne sme biti večja od 30 metrov.
3. Če je v razdalji 30 metrov več listov, jo povežemo na najbližji list.

3.2.3 Iskanje najkrajše poti

Če nam uspe poiskati, kam na omrežje se povežeta točki A in Z iz para, se lahko lotimo iskanja najkrajše poti. Soočamo se s problemom iskanja

najkrajše poti v neusmerjenem uteženem grafu. Povezave v našem primeru nimajo določene smeri oziroma so dvosmerne. Ceno povezav predstavlja razdalja, ki je shranjena kot lastnost vsake povezave. Pri iskanju najkrajših poti je še dodatna zahteva s strani strokovnjakov v podjetju: veljavne poti so samo tiste, ki so krajše od 8000 m.

Neo4j ponuja več vgrajenih funkcij za iskanje najkrajših poti. Za naš primer (ker iščemo v uteženem grafu) sta uporabni dve:

- z Dijkstrovim algoritmom in
- z algoritmom A*.

Dijkstrov algoritem se imenuje po znanstveniku Edsgerju Dijkstri, ki ga je leta 1956 razvil [20]. Rešuje problem iskanja najkrajše poti, pri čemer morajo biti cene povezav nenegativne vrednosti [21]. Algoritem iz podanega grafa, začetnega in končnega vozlišča, poišče najcenejšo pot. Deluje tako, da korak za korakom razvija drevo najkrajših poti, pri čemer se v drevo sprejemajo nova vozlišča, ki so sosedje že obstoječih vozlišč v drevesu. Večina sodobnih implementacij algoritma uporablja prioritetno vrsto za odločanje naslednjega vozlišča. Prioriteta se določi na podlagi najkrajše dolžine od začetnega vozlišča.

Algoritem A* je nadgradnja Dijkstrovega algoritma, ki so ga leta 1968 dopolnili Peter Hart, Nils Nilsson in Bertram Raphael na Raziskovalnem inštitutu Stanford [19]. A* je časovno manj zahteven, ker določi prioriteto glede na seštevek cene najkrajše poti do razvitega vozlišča in hevristične ocene preostanka poti do končnega vozlišča.

Po podrobnejšem pregledu Neo4j dokumentacije omenjenih funkcij smo ugotovili, da implementacija algoritma Dijkstra omogoča omejevanje cene (v našem primeru dolžine) poti, medtem ko algoritem A* te opcije nima vgrajene.

Točki A in Z se lahko nahajata v različnih grafih. Omrežje namreč ni v celoti povezano, zato tudi ni poti med vsemi kombinacijami vozlišč. V takšnem primeru bi algoritem A* preiskal celoten graf, v katerem se nahaja

točka A, in prišel do zaključka, da veljavna pot med A in Z ne obstaja. Algoritem Dijkstra bi prišel do istega zaključka, ko bi preiskal vse poti v grafu do točke A do dolžine 8000 m. Kadar gre za graf z razponom več 100 km z milijon povezavami je pričakovati, da bo Dijkstra znatno hitreje zaključil izvajanje.

Drugi aspekt je čas implementacije rešitve. Če se odločili za implementacijo z algoritmom A*, bi morali implementirati tudi hevristično oceno, na podlagi katere deluje algoritem. Na koncu bi bilo potrebno še preverjanje ustreznosti rešitve, ki bi izločila vse rešitve, katerih dolžina celotne poti presega 8000 m.

Zaradi zgoraj naštetih zapletov z algoritmom A* smo se odločili, da uporabimo algoritem Dijkstra.

3.2.4 Izpis poti

Zahteva za izhodne podatke naše aplikacije je, da se za pare točk, kjer smo našli veljavno pot, izpiše rešitev v CSV obliki, tiste, za katere ne obstaja veljavna pot, pa se izpusti. Izhodna datoteka mora vsebovati naslednje stolpce:

- `kabel_id` enolični identifikator para točk iz izvorne datoteke,
- `dolzina` dolžina najdene poti v metrih in
- `pot` zaporedno urejen seznam imen segmentov, ločen z vejicami.

Za ločilo med stolpci naj se uporablja podpičje. Primer glave in vrstice takšne datoteke:

```
kabel_id;dolzina;pot
1001;45;seg_1456,seg_25,seg_7817
```

Za potrebe testiranja aplikacije med razvojem smo se odločili za implementacijo še enega načina izpisa izhodnih podatkov – takšnega, ki ga bo enostavno uvoziti v QGIS in bo rešitev grafično vidna na zemljevidu. QGIS omogoča uvoz CSV datotek, kjer se v enem od stolpcev geometrijski element

predstavi z obliko Well-known text ali krajše WKT[3] in lastnosti elementa v ostalih stolpcih. Za testiranje bo zadostoval pregled, po katerih segmentih poteka pot in za kateri `kabel_id` gre. Ustrezen format izhodne datoteke CSV bo naslednji:

- `WKT`: črta v obliki `LINESTRING`, ki bo povezovala mejne točke vseh segmentov, in
- `kabel_id`: enolični identifikator para točk.

Primer glave in vrstice takšne datoteke:

```
WKT;kabel_id
LINESTRING(1 1, 2 5, 15 5, 30 10);cable_id_1001
```

[3]WKT je tekstovni označevalni jezik za opisovanje vektorskih geometrijskih elementov v prostorskih podatkih [18]. Na primer točko z WKT opišemo kot `POINT (30 10)`, črto kot `LINESTRING (30 10, 10 30, 40 40)`. WKT je uveljavljen standard.

Poglavje 4

Implementacija rešitve

4.1 Uvoz omrežja

Vhodni podatki v koraku za uvoz omrežja se nahajajo v obliki shapefile. Namen tega koraka je sestaviti graf, ki bo predstavljal omrežje. Za dosego tega cilja smo problem razdelili na dva podproblema. Prvi je branje podatkov iz shapefile oblike, drugi pa izgradnja grafa.

4.1.1 Branje iz shapefile datotek

Za branje podatkov iz oblike shapefile smo izbrali javansko odprtokodno knjižnico GeoTools, ki jo razvija in vzdržuje Open Source Geospatial Foundation [22]. V preteklosti smo že uspešno uporabljali knjižnice omenjene fundacije pri projektih z geografskimi podatki, kar je pripomoglo tudi k tokratni odločitvi.

Dostop do podatkov z GeoTools se je izkazal za zelo enostavnega. Podati je treba samo pot do datoteke s končnico `.shp`, nato pa razred za obdelavo datotek shapefile poskrbi za dostop do vseh sorodnih datotek, kjer so shranjeni atributi. Ta primerek razreda se uporabi za pridobivanje virov, nato pa se s pomočjo iteratorja sprehajamo po celotni zbirki.

Pridobljeni elementi so primerki razreda `SimpleFeature`, ki služi kot hramba lastnosti. Tipičen element iz oblike shapefile, bo imel lastnost z

imenom `GEOMETRY`, kjer bo shranjen geometrijski opis elementa, ter nabor vseh ostalih lastnosti, ki so vezane na element – v našem primeru je med njimi enolični identifikator segmenta.

Atribut `GEOMETRY` je mogoče v našem primeru pretvoriti (angl. cast) v razred `MultiLineString`, ker so vsi elementi v naši shapefile obliki črte. Razred `MultiLineString` omogoča pridobitev urejene zbirke koordinat črte z enim samim klicem funkcije. Enostavno se pridobi tudi dolžino celotne črte oz. segmenta, kar bomo uporabili kot ceno povezave pri izgradnji grafa. S tem so pridobljeni vsi potrebni podatki za izgradnjo grafa.

4.1.2 Izgradnja grafa

Grafno podatkovno bazo Neo4j je mogoče uporabljati v več načinih. Prvi način je samostojni strežnik, s katerim je mogoče komunicirati preko REST API-ja. Drugi je vgrajeni način (angl. embedded mode), ki omogoča dostopanje do grafne podatkovne baze neposredno preko klicev, ki nam jih omogoča knjižnica. Tretji način uporabe je prav tako vgrajeni način in je namenjen le začetnemu polnjenju baze. Uporaben je v primeru, ko moramo vstaviti velike količine podatkov. Ta način se imenuje paketno polnjenje (angl. batch insert). Performančno je za vstavljanje najučinkovitejši, saj se izogne transakcijam in drugim preverbam. Pri uporabi tega načina se je treba zavedati naslednjih stvari:

- namenjen je samo za začetno vstavljanje podatkov,
- ni nitno varen (angl. thread safe),
- ne uporablja transakcij,
- ob zaustavitvi (angl. shutdown) se bodo ponovno zgradili vsi indeksi in
- če proces zaustavitve ne bo uspešno dokončan po koncu vstavljanja, bodo datoteke podatkovne baze okvarjene.

```
BatchInserter batchInserter = BatchInserters.inserter ("/var/lib/pot_do_baze_na_disku/");
// ... koda za vstavljanje elementov ...
batchInserter.shutdown ( );
```

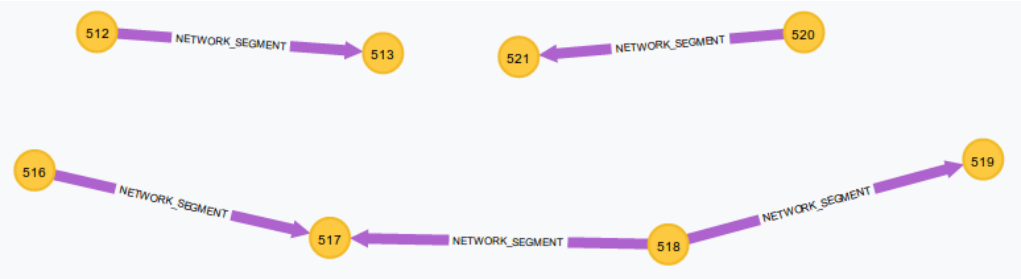
Slika 4.1: Primer ustvarjanja ter zaustavitve dostopa do baze na disku.

Zadnji omenjen način je za naš primer uporabe izgradnje grafa najbolj primeren, zato smo ga tudi uporabili.

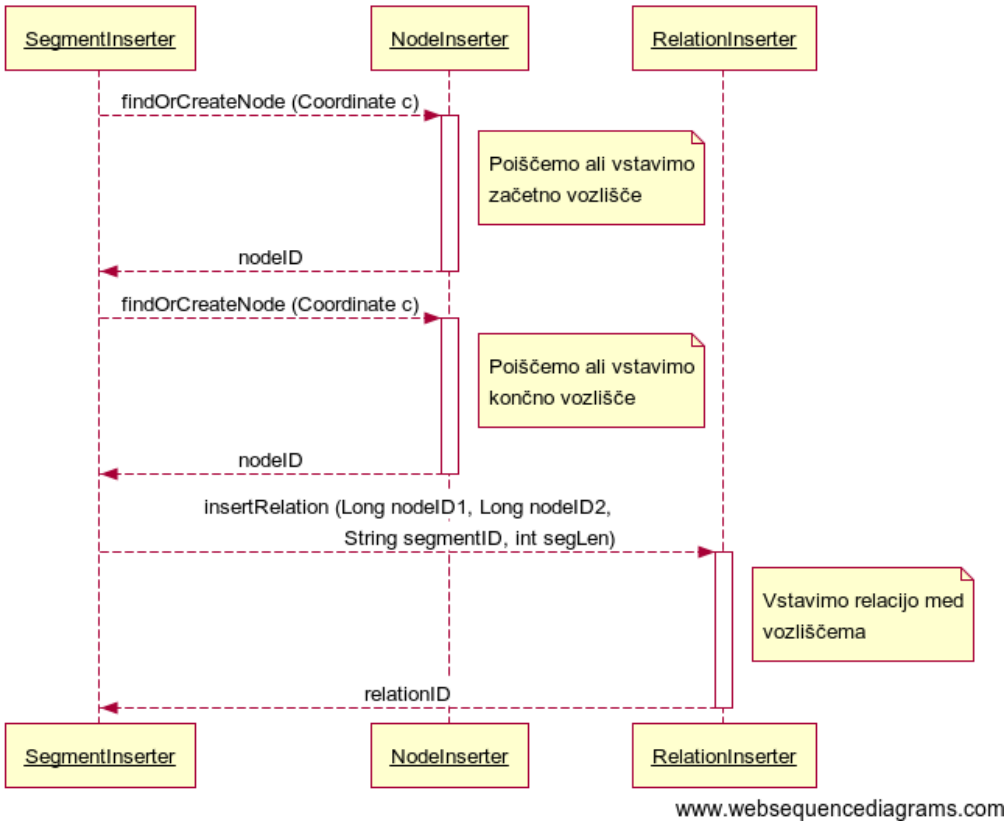
Za dostop do podatkovne baze v vgrajenem načinu izberemo primeren objekt, glede na način dostopa (v našem primeru `BatchInsert`). Kot parameter podamo pot do mape na disku, kjer se nahaja obstoječa baza oziroma kjer jo želimo ustvariti. Za ustvarjanje nove podatkovne baze niso potrebni dodatni ukazi. Datotečna struktura prazne podatkovne baze se zgradi samodejno ob prvem dostopu z izbranim dostopovnim objektom do mape, kjer baza še ne obstaja. Če baza v mapi že obstaja, se ob dostopu le preveri integriteta podatkov. Slika 4.1 prikazuje tipičen primer izvorne kode, kjer se ustvari dostop do podatkovne baze na disku in se na koncu zaustavi.

Dostop do podatkovne baze je zdaj urejen, naslednji korak je implementacija metode za vstavljanje segmentov, ki so bili pridobljeni z branjem iz datotek shapefile. Da lahko vstavimo segment, je treba najprej poiskati ali vstaviti začetno in končno vozlišče. Tem vozliščem bomo pripisali oznako `SEGMENT_POINT`. Poiskati zato, ker je možno, da vozlišče z enako koordinato že obstaja od takrat, ko smo vstavili sosednji segment. V takšnem primeru želimo uporabiti isto vozlišče, kot smo ga v sosednjem segmentu, saj le tako gradimo povezan graf[1]. Ko sta obe vozlišči vstavljeni, lahko med njima ustvarimo povezavo, zraven pa shranimo tudi lastnosti o enoličnem identifikatorju in dolžini segmenta. Slika 4.3 prikazuje diagram zaporedja za dodajanje segmenta, kot smo ga implementirali.

[1]Za lažjo predstavo slika 4.2 zgoraj prikazuje situacijo dveh nepovezanih segmentov, ter spodaj tri povezane segmente.



Slika 4.2: Povezani in nepovezani segmeti.



Slika 4.3: Diagram zaporedja dodajanja segmenta.

4.2 Povezovanje točk A in Z z grafom

Med načrtovanjem smo definirali pravilo, da moramo točke A in Z povezati na najbližji list grafa, ki pa ne sme biti oddaljen več kot 30 m. Za dosego tega cilja, se srečamo z dvema preprekama, in sicer z določitvijo listov in iskanjem lista v radiju 30 m.

4.2.1 Določitev listov grafa

List grafa je vozlišče, ki ima valenco oz. stopnjo 1. Valenca vozlišča določa, koliko povezav je vezanih na vozlišče. Vse liste grafa lahko torej določimo tako, da poiščemo vsa vozlišča, ki imajo samo eno povezavo. To lahko storimo s poizvedbenim jezikom Cypher z naslednjo poizvedbo:

```
1 MATCH (n:SEGMENT_POINT)-[s:NETWORK_SEGMENT]-()
2    WITH n,count(s) as rel_cnt
3    WHERE rel_cnt = 1
4 RETURN n AS leaf
```

V prvi vrstici poizvedbe definiramo vzorec iskanih povezav v grafu, ki nas zanima. To so vsa vozlišča z oznako `SEGMENT_POINT`. Označimo jih z `n`. Imeti morajo tudi tip povezave `NETWORK_SEGMENT` s katerim koli drugim vozliščem. V drugi vrstici dodamo vsakemu vozlišču `n`, ki ustreza vzorcu, še število povezav. V tretji vrstici filtriramo tiste, ki imajo število povezav enako 1 – ti so po definiciji listi. S četrto vrstico povemo prevajalniku, da nas kot rezultat poizvedbe zanimajo vsa ustrezna vozlišča `n`, stolpec pa naj se v rezultatih imenuje `leaf`.

4.2.2 Iskanje najbližjega lista v radiju 30 m

Iskanje najbližjega lista v radiju smo razčlenili na dva dela. Prvi del je Cypher poizvedba, ki vrne vse liste, v kvadratu 60 m x 60 m s središčem v iskani točki. Kvadrat zato, ker je poizvedba veliko enostavnejša in hitrejša, če iščemo znotraj kvadrata, kot če bi iskali znotraj kroga. Ker so merske enote v

našem uporabljenem koordinatnem sistemu metri, nam to precej poenostavi situacijo, saj zadošča že spodnja poizvedba:

```
MATCH (n:`LEAF_SEGMENT_POINT`)
WHERE n.x >= 121300 - 30
  AND n.x <= 121300 + 30
  AND n.y >= 35200 - 30
  AND n.y <= 35200 + 30
RETURN n AS leaf_candidates
```

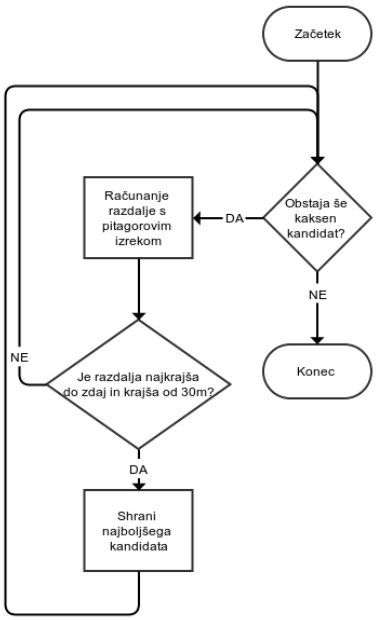
Poizvedba išče vsa vozlišča z oznako `LEAF_SEGMENT_POINT`, katerih lastnost `x` je znotraj intervala `[121300 - 30, 121300 + 30]` ter lastnost `y` znotraj intervala `[35200 - 30, 35200 + 30]`. Z drugimi besedami – iščemo vsa `LEAF_SEGMENT_POINT` vozlišča v kvadratu 60 m x 60 m okoli točke s koordinatama `x:121300, y:35200`. Pri uporabljeni poizvedbi je mogoče videti, da so se pri načrtovanju poizvedbenega jezika Cypher zgledovali tudi po SQL, saj je del stavka `WHERE` identičen, kot bi ga lahko uporabili v relacijskih bazah, ki uporabljajo poizvedbeni jezik SQL.

V drugem delu, med morebitno najdenimi vozlišči, poiščemo najbližjega, ki je v radiju 30 m. Za to moramo izračunati razdaljo vsakega vozlišča od točke, za kar lahko uporabimo pitagorov izrek. Diagram na sliki 4.4 prikazuje proces iskanja najbližjega kandidata.

4.3 Iskanje najkrajše poti

Če sta bila najdena lista, na katera se povežeta točki A in Z, lahko začnemo z iskanjem poti. V krajši raziskavi iz prejšnjega poglavja smo se odločili za razred Dijkstra iz knjižnice Neo4j. Preden se lotimo iskanja, je treba omenjenemu razredu določiti še sledeče, da pravilno konfiguriramo iskanje:

- začetno ceno poti,
- začetno vozlišče,



Slika 4.4: Diagram poteka iskanja najbližjega lista v radiju 30 m.

- končno vozlišče,
- evalvator cene,
- akumulator cene,
- primerjalnik (angl. comparator) cen,
- smer poti, na katero se mora algoritem omejiti, in
- seznam tipov povezav, na katere se mora algoritem omejiti.

Za konfiguracijo našega primera smo uporabili naslednje:

- začetna cena poti je 0,
- začetno vozlišče, ki smo ga našli za točko A,
- končno vozlišče, ki smo ga našli za točko Z,

- za evalvator cene smo uporabili vgrajeno implementacijo CommonEvaluators.doubleCostEvaluator, saj so vse naše cene podatkovnega tipa Double. Lastnost, ki jo je treba upoštevati za ceno, pa je shranjena pod imenom `SEGMENT_LEN`,
- za akumulator cene smo spet uporabili vgrajeno implementacijo `DoubleAdder`,
- za primerjalnik smo uporabili naravni primerjalnik knjižnice Apache commons,
- ker smer povezav pri nas ne igra vloge, smo izbrali `Direction.BOTH`,
- tip povezav, po katerih iščemo veljavno pot, pa smo omejili na `NETWORK_SEGMENT`.

Po tem, ko smo definirali vse zgoraj našteto, nam ostane samo še zadnja zahteva, da omejimo dolžino poti na 8000 m, kar storimo s klicem metode `limitMaxCostToTraverse`, kot parameter pa podamo 8000.

S tem je naš primerek razreda za iskanje poti pripravljen. Ker nas zanima seznam povezav, po katerih pridemo od točke A do točke Z, uporabimo klic metode `getPathAsRelationships`, ki vrne `List<Relationship>`. Če ta seznam ni ničen (angl. null), pomeni da smo uspešno našli pot in lahko izvedemo še zadnji korak v procesu, to je izpis poti.

4.4 Izpis poti

Za izpis poti smo se odločili, da implementiramo dve rešitvi. Skupno obema rešitvama je, da bosta za izhodne podatke uporabljali obliko CSV ter da bo vsaka vrstica predstavljala opis rešitve za en par točk. Zaradi teh skupnih točk smo ustvarili abstraktni nadrazred, ki poskrbi za vso kodo, ki je potrebna za ustvarjanje in zaključevanje datotek CSV. Za delo z obliko CSV smo uporabili odprtokodno knjižnico opencsv. Podrazredoma tako ostane

samo implementacija metode, ki na podlagi dobljene poti izpiše primerno obliko vrstice datoteke CSV.

4.4.1 Izpis z identifikatorji segmentov poti

Vhodni parametri metode so trije, in sicer:

- razred za opis para točk A in Z,
- urejen seznam povezav, preko katerih poteka najdena pot, in
- dolžina poti v metrih.

Kot je bilo določeno v prejšnjem poglavju, morajo vsebovati izhodni podatki naslednje stolpce: `kabel_id`, `dolzina` in `pot`. Izpis vrednosti za `kabel_id` in `dolzina` je iz dobljenih vhodnih podatkov trivialno opravilo. Za izpis segmentov poti pa je potrebna iteracija čez seznam povezav z izpisom lastnosti povezave, ki določa enolično identifikacijo segmenta.

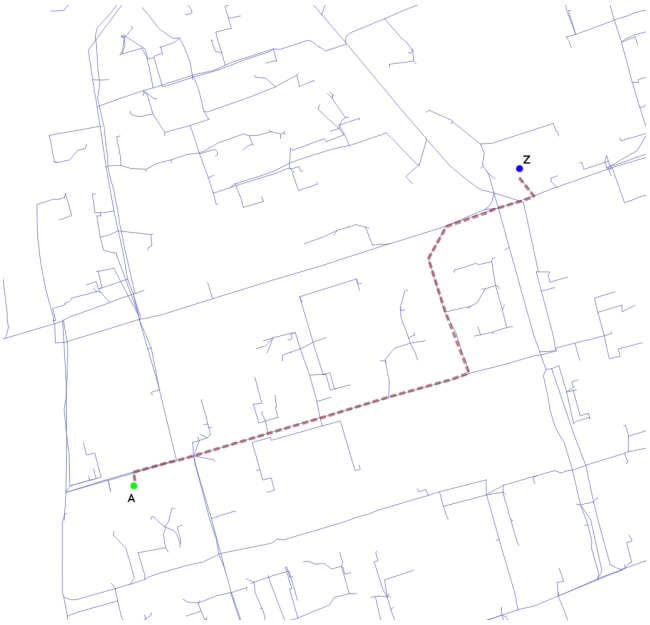
Oblika izpisa datoteke CSV je enaka, kot smo jo načrtovali v prejšnjem poglavju pri izpisu poti.

4.4.2 Izpis v WKT obliki

Implementacija te metode se ne razlikuje bistveno od prej opisane metode, le da dobimo kot vhodni parameter v tem primeru seznam vozlišč namesto seznama povezav. Tokrat iteriramo preko seznama vozlišč, kjer izpisujemo lastnosti `x` in `y` vozlišč, ki predstavljajo stične točke segmentov. Končni rezultat je črta v WKT obliki LINESTRING.

Tudi v tem primeru se oblika izpisa povsem sklada z načrtovano obliko iz prejšnjega poglavja.

Slika 4.5 prikazuje najdeno pot med točkama A in Z, ki smo jo uvozili iz datoteke CSV v aplikacijo QGIS. Debelejša prekinjena črta predstavlja pot, tanjše polne črte pa predstavljajo omrežje.



Slika 4.5: Najdena pot od točke A(centrale) do točke Z(uporabnika).

št. vseh preiskanih parov	84357
št. najdenih rešitev	60746
odstotek najdenih rešitev	72%

Tabela 4.1: Uspešnost iskanja rešitev.

4.5 Povzetek rešitve

Končna rešitev se je izkazala zelo dobro. S stališča programske kode je rešitev zelo pregledna, razumljiva in primerna za morebitne nadgradnje.

Iz tabele 4.1 je razvidno, da je uspešnost iskanja rešitve 72-odstotna. To je bilo glede na pomanjkljive vhodne podatke tudi pričakovan izid. Čas izvajanja, ki je prikazan v tabeli 4.2, je za trenutne potrebe povsem zadovoljiv.

Uporabniki so bili nad rezultati zelo navdušeni, saj jim je naša rešitev omogočila delo na novih področjih, ki so bila prej nedosegljiva. Brez naše rešitve ne bi mogli izvajati določenih analiz, saj bi morali iskati poti ročno, kar pa s takšno količino podatkov realno sploh ni izvedljivo. Točnost po-

čas izvajanja za 84357 parov	1.350 s
povprečni čas iskanja rešitve	16 ms

Tabela 4.2: Meritve časov izvajanja na povprečni strojni opremi za domačo uporabo.

datkov se je izkazala za povsem zadovoljivo. Edina želja, ki so jo izrazili, je bila, da bi bilo najdenih več rešitev. Za to bi bilo treba pridobiti popolnejše podatke o omrežju in lokacijah uporabnikov, kar pa je oboje izven meja našega sistema. V mejah našega sistema bi lahko pripomogli k izboljšanju števila najdenih rešitev, če bi v sodelovanju s strokovnjaki na področju telekomunikacij določili še dodatna hevristična pravila, s katerimi bi dopolnili pomanjkljive vhodne podatke.

Poglavje 5

Zaključek

V sklopu tega diplomskega dela je bila izdelana delujoča aplikacija za iskanje poti v telekomunikacijskem omrežju. Uporabili smo sodobne metodologije razvoja aplikacij. Zajeli smo zahteve uporabnikov, naredili načrt aplikacije, poiskali primerna orodja, implementirali aplikacijo in jo na koncu testirali ter dostavili uporabnikom s primernimi navodili za uporabo, tako da jo zdaj uporabljajo.

Aplikacija se je v praksi izkazala za zelo uporabno. Presegla je celo pričakovanja uporabnikov. Zaradi njene uporabnosti je zelo verjetno, da bo produkt ostal živ in doživel tudi izboljšave in nadgradnje.

Možnosti za nadgradnjo sistema je več. Med najbolj očitnimi je dopolnitev hevrističnih pravil z namenom izboljšanja odstotka najdenih rešitev. Trenutno je mogoče aplikacijo pognati zgolj iz ukazne vrstice, lahko pa bi dodatno naredili tudi grafični vmesnik. Ne nazadnje bi lahko tudi pospešili izvajanje z vzporednim izvajanjem več niti, učinkovitejšo uporabo indeksov pri iskanju listov in z uporabo učinkovitejšega algoritma za iskanje najkrajše poti.

Čeprav je bila aplikacija razvita posebej za potrebe telekomunikacij, bi bilo mogoče to aplikacijo brez ali z manjšimi modifikacijami uporabiti tudi na drugih podobnih področjih, na primer v omrežju dobave električne energije.

Literatura

[1] S. Drobne, Uvod v geografske informacijske sisteme in prostorske analize. Dostopno na: http://www.km.fgg.uni-lj.si/PREDMETI/TIUS/data/GIS/ORG-%20GIS%20in%20PA.pdf, 2014

[2] Ian Robinson, Jim Webber, and Emil Eifrem *Graph Databases*, O'Reilly, 2013.

[3] Trendi popularnosti grafnih podatkovnih baz. Dostopno na: http://db-engines.com/en/ranking_trend/graph+dbms, 2014

[4] Trendi popularnosti iskalnega pojma "graph database". Dostopno na: http://www.google.com/trends/explore#q=%22graph%20database%22&date=1%2F2008%2075m&cmpt=q, 2014

[5] Neo4j wiki. Dostopno na: http://en.wikipedia.org/wiki/Neo4j, 2014

[6] Cypher Query Language. Dostopno na: http://docs.neo4j.org/chunked/stable/cypher-query-lang.html, 2014

[7] Graf. Dostopno na: http://wiki.fmf.uni-lj.si/wiki/Graf, 2014

[8] Graph databases: an overview. Dostopno na: http://blog.octo.com/en/graph-databases-an-overview/, 2014

[9] Neo4j. Dostopno na: http://www.neo4j.org/, 2014

[10] Cypher introduction. Dostopno na: http://docs.neo4j.org/chunked/stable/cypher-introduction.html, 2014

[11] NoSQL. Dostopno na: http://en.wikipedia.org/wiki/NoSQL, 2014

[12] Google maps. Dostopno na: http://maps.google.com, 2014

[13] Natural Earth. Dostopno na: http://www.naturalearthdata.com/, 2014

[14] ESRI. Dostopno na: http://www.esri.com/, 2014

[15] DBase. Dostopno na: http://www.dbase.com, 2014

[16] ESRI Shapefile Technical Description, ESRI, 1998

[17] QGIS. Dostopno na: http://qgis.org/, 2014

[18] Open GIS Consortium, Inc., OpenGIS Simple Features Specification For SQL, 1999

[19] Peter E. Hart et al., A Formal Basis for the Heuristic Determination of Minimum Cost Paths, Systems Science and Cybernetics, IEEE, 1968

[20] J. Russell in R. Cohn, Dijkstra's Algorithm, Book on Demand, 2012

[21] Thomas H. Cormen et al., Introduction to algorithms, 3rd edition, The MIT Press, 2009

[22] GeoTools. Dostopno na: http://www.geotools.org/, 2014